# Implementasi Fungsi *Hash* untuk Pertukaran Data pada Telepon Seluler

Juliana Amytianty Kombaitan - 13507068[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
[1]*if17068@students.if.itb.ac.id*

***Abstrak*— Pada zaman modern ini, hampir seluruh lapisan masyarakat memiliki telepon seluler atau yang populer disebut *handphone* (HP). Dengan berkembangnya teknologi, pertukaran data semakin mudah dilakukan melaui telepon seluler, apalagi telepon seluler yang berbasis multimedia. Data yang dikirim dapat berupa teks (SMS atau *notes*), foto, audio, maupun video. Namun, semakin canggih teknologi, semakin canggih juga teknologi penyadapan data. Hal ini menyebabkan makin renggangnya kepercayaan mengenai otentikasi dan integritas data yang dipertukarkan. Untuk mengatasi masalah ini, fungsi *hash* dapat menjadi salah satu jalan keluar. Fungsi *hash* adalah fungsi satu arah yang mengubah suatu data menjadi suatu *hash* atau *message digest*. Nilai *hash* inilah yang akan digunakan untuk mengecek apakah data yang dipertukarkan dimodifikasi atau tidak, atau untuk mengecek apakah datanya berasal dari orang yang benar.**

***Kata Kunci*—fungsi *hash*, integritas data, otentikasi data, telepon seluler**

## I. PENDAHULUAN

Perkembangan teknologi yang semakin berkembang sekarang mendorongnya naiknya gaya hidup masyarakat, salah satunya dalam bidang komunikasi. Pada era modern ini, terdapat lebih banyak media yang mendukung komunikasi dibandingkan zaman dahulu. Salah satunya adalah dalam hal bertelepon. Zaman sekarang, **telepon seluler** atau *handphone* sudah tidak asing lagi terdengar di telinga kita. Kurang lebih satu dekade yang lalu, telepon seluler adalah barang langka atau barang mewah bagi masyarakat. Orang biasanya memakai telepon rumah atau memakai surat pos. Namun di zaman sekarang, telepon seluler adalah barang yang menjadi kebutuhan banyak orang. Hampir semua kalangan masyarakat menggunakan telepon seluler, mulai dari sopir angkot samapi pejabat tinggi. Telepon seluler sudah menjadi *gadget* yang tidak dapat dipisahkan dari sisi kehidupan masyarakat.

Dengan semakin banyaknya penggunaan telepon seluler, masyarakat sekarang lebih senang mengelola data melalui telepon seluler. Semua jenis data, baik data teks maupun multimedia dapat diakses lewat telepon seluler. Dengan demikian pertukaran data, baik pengiriman maupun penerimaan data dapat dilakukan melalui media telepon seluler. Segala bentuk pertukaran menjadi mudah dilakukan lewat SMS (*Short Messaging Service*) yang berbasis teks, MMS (*Multimedia Messaging Service*). Kedua jenis pertukaran data ini dilakukan melalui operator. Selain itu ada juga pertukaran data lewat menggunakan sinar infra merah, *bluetooth,* kabel data, atau kartu memori. Untuk yang berbasis internet, pertukaran data didukung oleh GPRS (*General packet Radio Service*), Wi-Fi (*Wireless-Fidelity*), 3G, dan lain-lain. Semua perangkat yang mendukung pertukaran data tersebut marak digunakan masyarakat pada zaman sekarang ini.

**Gambar 1 Perkembangan telepon seluler dari zaman ke zaman**

Semakin banyak perangkat yang digunakan untuk pertukaran data, semakin banyak penyusup atau penggangu dalam proses pengiriman data. Beberapa gangguan yang berkaitan dengan keamanan proses pertukaran data antara lain:

- Data rahasia dapat dibuka dan diketahui isinya oleh pihak ketiga
- Data dimodifikasi oleh pihak ketiga, sehingga data yang sampai ke penerima berbeda dengan data asli yang dikirim
- Data dimanipulasi sehingga informasi pengirim dalam data bukan jati diri pengirim sebenarnya
- Terjadi penyangkalan dari pengirim bahwa ia telah

mengirim data kepada penerima.

Gangguan tersebut dapat terjadi melalui koneksi yang disebutkan di atas. Misalnya jika sesorang mengirim SMS yang berbasis teks, teks yang dikirim benar-benar berupa teks saja, sehingga memungkinkan pihak ketiga untuk memodifikasi atau merusak isi pesan tersebut. Contoh lain, misalkan pertukaran data menggunakan *bluetooth.* Jika pengirim tidak hati-hati data yang dikirim dapat dirusak oleh pihak ketiga, misalnya virus yang disebarkan melalui perangkat *bluetooth.*

## II. TANDA TANGAN DIGITAL

Untuk mengatasi masalah ketidakamanan pertukaran data pada telepon seluler, kita dapat menggunakan tanda tangan digital. Tanda tangan digital adalah tanda tangan untuk data digital. Sama seperti data tertulis, tanda tangan digital merepresentasikan keunikan pemilik dat tersebut. Tanda tangan digital adalah nilai kriptografis yang bergantung pada isi pesan dan kunci (tanda tangannya). Tanda tangan digital selalu berbeda-beda antara data yang satu dengan data yang lain.

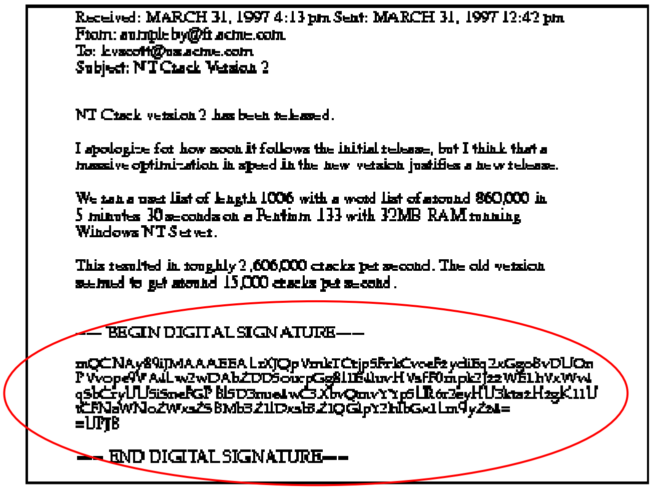
Dalam ilmu kriptografi, terdapat empat aspek yang ditangani oleh kriptografi, yaitu kerahasiaan pesan (*secrecy*), otentikasi (authentication), keaslian pesan (*integrity*), dan anti penyangkalan (*nonrepudiation*). Dari keempat aspek tersebut, aspek yang ditangani tanda tangan digital adalah aspek otentikasi, keaslian pesan, dan anti penyangkalan. Untuk aspek otentikasi, tanda tangan digital menunjukkan keunikan identitas pengirim atau pemilik data, karena setiap dokumen yang berbeda memiliki tanda tangan digital yang berbeda juga. Untuk aspek keaslian pesan, tanda tangan digital akan mengidentifikasi jika terjadi perubahan isi data. Hal ini mungkin terjadi karena perubahan sedikit saja pada isi data membuat nilai tanda tangan digital berubah secara signifikan. Sedangkan untuk aspek anti penyangkalan, tanda tangan digital mengidentifikasi pemilik data, sehingga kecil sekali kemungkinannya data tersebut bukan kepunyaan orang yang tanda tangan digitalnya sama dengan yang bersangkutan. Untuk alasan ini, orang yang bersangkutan tidak dapat memungkiri data yang dimilikinya, karena terdapat buktinya, yaitu tanda tangan digitalnya.

Karakteristik tanda tangan digital adalah sebagai berikut:

- Tanda-tangan adalah bukti yang otentik.
- Tanda tangan tidak dapat dilupakan.
- Tanda-tangan tidak dapat dipindah untuk digunakan ulang.
- Dokumen yang telah ditandatangani tidak dapat diubah.
- Tanda-tangan tidak dapat disangkal (*repudiation*).

Terdapat beberapa cara yang dilakukan untuk membuat tanda tangan digital, antara lain:

1. Mengenkripsi pesan
   Pesan yang dienkripsi menyatakan bahwa pesan tersebut sudah ditandantangani. Jadi yang berupa tanda tangan digital dari data tersebut adalah cipherteksnya.
2. Menggunakan fungsi *hash*
   Tanda tangan digital dibuat menggunakan fungsi *hash* yang menghasilkan *hash / message digest. Message digest* inilah yang menjadi tanda tangan digitalnya.



Received: MARCH 31, 1997 4:13 pm Sent: MARCH 31, 1997 12:42 pm
From: aumpleby@ft.acme.com
To: kvscott@ss.acme.com
Subject: NT Crack Version 2

NT Crack version 2 has been released.

I apologize for how soon it follows the initial release, but I think that a massive optimization in speed in the new version justifies a new release.

We ran a user list of length 1006 with a word list of around 860,000 in 5 minutes 30 seconds on a Pentium 133 with 32MB RAM running Windows NT Server.

This resulted in roughly 2,606,000 cracks per second. The old version seemed to get around 15,000 cracks per second.

—— BEGIN DIGITAL SIGNATURE——

——— END DIGITAL SIGNATURE———

**Gambar 2 Dokumen dengan tanda tangan digital**

Tanda tangan digital biasanya digabungkan (*append*) pada data yang bersangkutan. Ketika data ini dikirim dan sampai kepada penerima, penerima menguji apakah tanda tangannya bersesuaian dengan data yang dikirimkan. Jika tanda tangannya cocok, berarti data tidak mengalami gangguan keamanan dalam proses pengiriman.

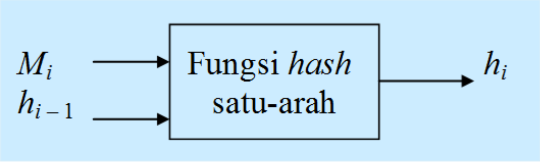
## III. FUNGSI *HASH*

Fungsi *hash* yang akan dibahas adalah fungsi *hash* kriptografi, yaitu fungsi *hash* satu arah. Fungsi *hash* adalah fungsi komputasi yang efisien untuk memetakan string biner dengan panjang yang tidak tetap untuk string biner yang panjangnya tetap, yang disebut nilai *hash* atau *message digest.* Untuk setiap dokumen atau pesan, nilai *hash* yang dihasilkan berbeda-beda.

Penggunaan kriptografi yang paling umum untuk fungsi *hash* adalah untuk tanda tangan digital dan integritas data. Dengan tanda tangan digital, pesan yang panjang biasanya diubah mejadi nilai *hash.* Pihak yang menerima pesan kemudian melakukan fungsi *hash* pada pesan, dan memastikan tanda tangan yang diterima sama dengan nilai *hash*-nya. Hal ini menghemat waktu dan tempat dibandingkan dengan membandingkan isi pesan secara langsung.

### *A. Cara Kerja Fungsi Hash Satu Arah*

Fungsi *hash* satu arah dapat diterapkan pada blok data berukuran bebas, sampai batas maksimal masing-masing variasi fungsi *hash.* Skema fungsi *hash* adalah sebagai berikut.



**Gambar 3 Skema fungsi *hash* satu arah**

Pada gambar di atas, M menyatakan satu blok pesan dan h menyatakan satu blok nilai *hash*. Masukan fungsi *hash* adalah blok pesan (M) dan keluaran *hash* dari blok pesan sebelumnya. Nilai *hash* yang dikeluarkan memiliki panjang tetap. Fungsi *hash* ini dikatakan satu arah karena untuk setiap h yang dihasilkan, tidak mungkin dikembalikan nilai x sedemikian sehingga H(x) = h. Untuk setiap x yang siberikan, tidak mungkin mencari pasangan x dan y sedemikian sehingga H(x) = H(y).

Fungsi *hash* cukup dapat diandalkan sebagai tanda tangan digital, karena fungsi *hash* sangat peka terhadap perubahan satu bit pada pesan. Jika pesan berubah 1 bit saja, perubahan yang terjadi pada nilai *hash* sangat signifikan.

## *B. Variasi Fungsi Hash*

Terdapat banyak fungsi *hash* yang sudah diimplementasikan. Fungsi *hash* mengalami perbaikan dan perkembangan seiring perkembangan zaman.

Beberapa variasi fungsi *hash* antara lain:

1. *Secure Hash Algorithm* (SHA)

   SHA bisa dikatakan algoritma yang aman karena algoritma ini dirancang sedemikian rupa sehingga secara komputasi tidak mungkin menemukan pesan yang berkoresponden dengan nilai *hash*. Algoritma ini menerima masukan dengan ukuran maksimum $2^{64}$ bit dan menghasilkan nilai *hash* sepanjang 160 bit. Algoritma SHA sudah berkembang sampai 3 generasi, yaitu:

   - SHA-1, yang merupakan fungsi *hash* 160 bit asli. SHA-1 adalah SHA yang paling banyak digunakan dalam aplikasi dan protokol.
   - SHA-2, memiliki ukuran blok yang berbeda-beda, yang dikenal sebagai SHA-256 (word 32 bit) dan SHA-512 (word 64 bit).
   - SHA-3 masih dalam tahap pengembangan

**Tabel 1 Pseudocode SHA-1**

```
Note 1: All variables are unsigned 32 bits and wrap modulo 2^32 when calculating
Note 2: All constants in this pseudo code are in big endian.
        Within each word, the most significant byte is stored in the leftmost byte position

Initialize variables:
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

Pre-processing:
append the bit '1' to the message
append 0 ≤ k < 512 bits '0', so that the resulting message length (in bits)
   is congruent to 448 ≡ -64 (mod 512)
append length of message (before pre-processing), in bits, as 64-bit big-endian integer

Process the message in successive 512-bit chunks:
break message into 512-bit chunks
for each chunk
    break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15

    Extend the sixteen 32-bit words into eighty 32-bit words:
    for i from 16 to 79
        w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1

    Initialize hash value for this chunk:
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4

    Main loop:
    for i from 0 to 79
        if 0 ≤ i ≤ 19 then
            f = (b and c) or ((not b) and d)
            k = 0x5A827999
        else if 20 ≤ i ≤ 39
            f = b xor c xor d
            k = 0x6ED9EBA1
        else if 40 ≤ i ≤ 59
            f = (b and c) or (b and d) or (c and d)
            k = 0x8F1BBCDC
        else if 60 ≤ i ≤ 79
            f = b xor c xor d
            k = 0xCA62C1D6

        temp = (a leftrotate 5) + f + e + k + w[i]
        e = d
        d = c
        c = b leftrotate 30
        b = a
        a = temp

    Add this chunk's hash to result so far:
    h0 = h0 + a
    h1 = h1 + b
    h2 = h2 + c
    h3 = h3 + d
    h4 = h4 + e

Produce the final hash value (big-endian):
digest = hash = h0 append h1 append h2 append h3 append h4
```

2. *Message Digest* 5 (MD5)

   MD5 adalah algoritma yang dibuat oleh Ron Rivest. MD5 merupakan perbaikan dari MD4 setelah MD4 dapat diserang. Algoritma MD5 menerima masukan berupa pesan dengan ukuran sembarang dan menghasilkan message digest yang panjangnya 128 bit.

**Tabel 2 Pseudocode MD5**

```
//Note: All variables are unsigned 32 bits and wrap modulo 2^32 when calculating
var int[64] r, k

//r specifies the per-round shift amounts
r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}

//Use binary integer part of the sines of integers (Radians) as constants:
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × (2 pow 32))
```

```
//Initialize variables:
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

//Pre-processing:
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod
512)
append  bit  /*  bit,  not  byte  */  length  of  unpadded
message as 64-bit little-endian integer to message

//Process the message in successive 512-bit chunks:
for each 512-bit chunk of message
    break chunk into sixteen 32-bit little-endian words
w[i], 0 ≤ i ≤ 15

    //Initialize hash value for this chunk:
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3

    //Main loop:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f := (b and c) or ((not b) and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d) and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16

        temp := d
        d := c
        c := b
        b  :=  b  +  leftrotate((a  +  f  +  k[i]  +  w[g])  ,
r[i])
        a := temp

    //Add this chunk's hash to result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d

var  int  digest  :=  h0  append  h1  append  h2  append  h3
//(expressed as little-endian)

//leftrotate function definition
  leftrotate (x, c)
      return (x << c) or (x >> (32-c));
```

**Tabel 3 Perbandingan beberapa fungsi *hash***

| **Algoritma** | **Ukuran *message digest* (bit)** | **Ukuran blok pesan** | **Kolisi** |
|---|---|---|---|
| *MD2* | 128 | 128 | Ya |
| *MD4* | 128 | 512 | Hampir |
| *MD5* | 128 | 512 | Ya |
| *RIPEMD* | 128 | 512 | Ya |
| *RIPEMD-128/256* | 128/256 | 512 | Tidak |
| *RIPEMD-160/320* | 160/320 | 512 | Tidak |
| *SHA-0* | 160 | 512 | Ya |
| *SHA-1* | 160 | 512 | Ada cacat |
| *SHA-256/224* | 256/224 | 512 | Tidak |
| *SHA-512/384* | 512/384 | 1024 | Tidak |
| *WHIRLPOOL* | 512 | 512 | Tidak |

## IV. SOLUSI MASALAH PERTUKARAN DATA PADA TELEPON SELULER

Untuk mengatasi masalah pertukaran data pada telepon seluler, fungsi *hash* satu arah dapat digunakan untuk tanda tangan digital. Ketika seseorang akan mengirim data yang penting kepada orang lain, ia dapat membubuhkan tanda tangan digital ke dalam data tersebut. Setelah data sampai kepada penerima, data dapat diuji keasliannya dengan membandingkan nilai *hash* pesan dengan tanda tangan digitalnya. Supaya hal ini dapat terjadi, harus ada persetujuan antara kedua pihak terlebih dahulu jenis fungsi *hash* apa yang akan dipakai. Selain itu, kedua pihak harus memiliki aplikasi tanda tangan digital yang sama, minimal terdapat pembangkit nilai *hash* yang disepakati bersama.

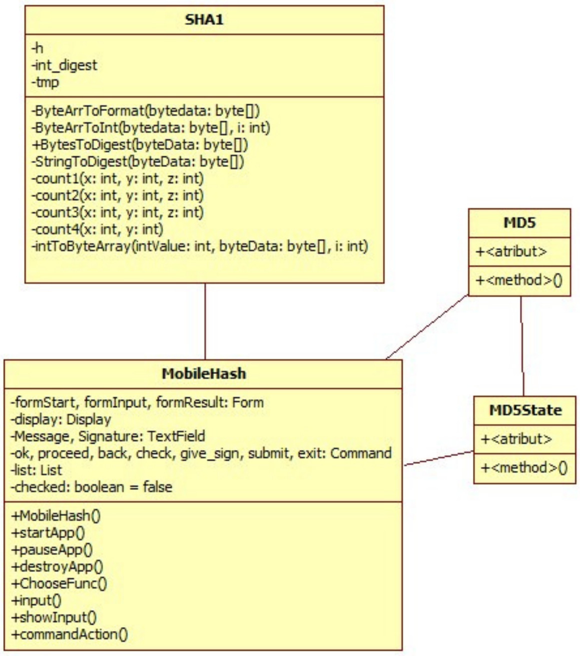
### *A. Implementasi Aplikasi Fungsi Hash pada Telepon Seluler*

Untuk membuat aplikasi pada telepon seluler, kita harus mengerti bahasa pemrograman yang digunakan pada telepon seluler. Bagi sebagian besar telepon seluler, aplikasi yang biasa digunakan menggunakan bahasa Java (J2ME). J2ME adalah versi dari bahasa pemrograman Java yang merupakan singkatan dari Java 2 Micro Edition. J2ME dirancang dengan keterbatasan memori dan prosesor perangkat elektronik kecil, seperti ponsel dan asisten digital pribadi (PDA). J2ME sebenarnya sama dengan pemrograman menggunakan java sendiri, hanya saja dalam J2ME ada beberapa fungsionalitas yang ditambah dan dikurangi dan di sesuaikan untuk pemrograman perangkat *mobile*.Untuk telepon seluler yang memiliki sistem operasi Symbian, aplikasi yang digunakan biasanya menggunakan bahasa C++.

Pada rancangan implementasi fungsi *hash* ini, bahasa yang digunakan adalah bahasa Java, dengan menggunakan NetBeans IDE 6.8. Proyek yang dibuat pada NetBeans adalah Java ME (Mobile Application). Aplikasi ini diberi nama Mobile Hash. Fungsi *hash* yang akan diimplementasikan adalah fungsi *hash* SHA-1 dan MD5. Jadi pengguna dapat memilih salah satu dari kedua fungsi *hash* tersebut. Sedangkan data yang dapat digunakan untuk Mobile Hash adalah data teks, sehingga data yang dapat ditandatangani dapat berupa SMS, memo, buku telepon (*phonebook*), kegiatan kalender, *to do list,* dan data lainnya yang berbasis teks. Sedangkan kemampuan dari paket perangkat yang akan anda jadikan target aplikasi adalah CLDC 1.1 dan MIDP 2.0.

Konsep program J2ME adalah sebagai berikut. Dalam J2ME kita harus akan membuat main class turunan dari class MIDlet, main class turunan dari MIDlet tersebut yang nantinya akan dipanggil pertama kali saat aplikasi kita berjalan di telepon seluler, dalam main class tersebut juga ada 3 method yang nantinya berfungsi sebagai *trigger even* dari ponsel kita, yaitu: **starApp()** yang merupakan method yang dipanggil apabila aplikasi kita pertama kali jalan; **pauseApp()**, yaitu method yang dipanggil apabila pengguna ponsel mem-*pause* aplikasi; dan **destroyApp()**, yaitu method yang dipanggil apabila pengguna ponsel menutup aplikasi.

Sebagai desain awal, diagram kelas Mobile Hash adalah seperti di bawah ini.



**Gambar 4 Diagram kelas Mobile Hash**

*Source code* untuk mengimplementasikan program utama adalah sebagai berikut.

**Tabel 4 Program Utama (MobileHash.java)**

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class MobileHash extends MIDlet implements
CommandListener{
  private Form formStart, formInput, formResult;
  private Display display;
  private TextField message, signature;
  private Command ok, proceed, back, check, give_sign,
submit, exit;
  private List list;
  private boolean checked = false;

  public MobileHash(){
    message = new TextField("Message:", "", 1000,
TextField.ANY);
    signature = new TextField("Signature:", "", 50,
TextField.ANY);
    ok = new Command("Options", Command.OK, 1);
    proceed = new Command("Proceed", Command.OK, 1);
    submit = new Command("Submit", Command.SCREEN, 1);
    back = new Command("Back",Command.BACK, 3);
    check = new Command("Check Authentication",
Command.OK, 4);
    give_sign = new Command("Give Signature",
Command.SCREEN, 5);
    exit = new Command("Exit", Command.EXIT, 6);
    list = new List("Hash Function", List.EXCLUSIVE);
  }

  public void startApp(){
    display = Display.getDisplay(this);
    formStart = new Form("Digital Signature");
    formStart.append("WELCOME TO DIGITAL SIGNATURE");
    formStart.addCommand(proceed);
    formStart.addCommand(exit);
    formStart.setCommandListener(this);
    display.setCurrent(formStart);
  }

  public void ChooseFunc() {
    display = Display.getDisplay(this);
    list.append("SHA-1", null);
    list.append("MD5", null);
    list.addCommand(submit);
    list.setCommandListener(this);
    display.setCurrent(list);
  }

  public void input() {
    display = Display.getDisplay(this);
    formInput = new Form("Digital Signature");
    formInput.append(message);
    formInput.append(signature);
    formInput.addCommand(back);
    formInput.addCommand(check);
    formInput.addCommand(give_sign);
    formInput.addCommand(exit);
    formInput.setCommandListener(this);
    display.setCurrent(formInput);
  }

  public void showInput(){
    display = Display.getDisplay(this);
    String m = message.getString();
    String s = signature.getString();
    formResult = new Form("RESULT");
    // proses pengecekan
    String digest = new
SHA1().StringToDigest(m.getBytes());
        System.out.println(digest);
        if
(list.getString(list.getSelectedIndex()).equals("SHA1")
){
            if (s.equals(digest)){
                formResult.append("Authentication was
successful!");
            }
            else if (s.length() == 0){
                formResult.append("Signature field is
empty!");
            }
            else {
                formResult.append("Authentication was
not successful! There's modification!");
            }
        }
        else if
(list.getString(list.getSelectedIndex()).equals("MD5"))
{
            MD5 md5 = new MD5(m.getBytes());
            byte[] result = md5.doFinal();
            String hashResult = md5.toHex(result);
            System.out.println(hashResult);
            if (s.equals(hashResult)){
                formResult.append("Authentication was
successful!");
            }
            else if (s.length() == 0){
                formResult.append("Signature field is
empty!");
            }
            else {
                formResult.append("Authentication was
not successful! There's modification!");
            }
        }
    formResult.addCommand(back);
    formResult.addCommand(exit);
    formResult.setCommandListener(this);
    display.setCurrent(formResult);
  }

  public void commandAction(Command c, Displayable d) {
    String label = c.getLabel();
    if(label.equals("Check Authentication")){
        showInput();
    }
    else if (label.equals("Proceed")){
        if (checked == false){
            ChooseFunc();
        }
        else {
            display.setCurrent(list);
        }
    }
    else if (label.equals("Back")){
```

```
        display.setCurrent(formInput);
        checked = true;
    }
    else if (label.equals("Submit")){
        if (checked == false){
            input();
        }
        else {
            display.setCurrent(formInput);
        }
    }
    else if (label.equals("Give Signature")){
        if
(list.getString(list.getSelectedIndex()).equals("SHA1")
){
            signature.setString(new
SHA1().StringToDigest(message.getString().getBytes()));
            }
        else                                            if
(list.getString(list.getSelectedIndex()).equals("MD5"))
{
            MD5           md5           =           new
MD5(message.getString().getBytes());
            byte[] result = md5.doFinal();
            signature.setString(md5.toHex(result));
            }
        display.setCurrent(formInput);
        checked = true;
    }
  }

  public void pauseApp(){

  }

  public void destroyApp(boolean destroy){
    notifyDestroyed();
  }
}
```

Pada program utama ini, digunakan 3 buah *form* dan 1 buah *list*. Form dan list adalah bagian dari tampilan pada J2ME. Form yang dibuat adalah formStart, formInput, dan formResult. Form formStart berisi tampilan awal Mobile Hash, formInput berisi masukan pesan dan pembuatan tanda tangan, sedangkan formResult berisi hasil pengujian otentikasi data. List digunakan untuk memilih salah satu fungsi *hash* (SHA-1 atau MD5).

Sedangkan untuk implementasi fungsi *hash* SHA 1, *source code* Java-nya adalah sebagai berikut.

**Tabel 5 Fungsi *Hash* SHA1 (SHA1.java)**

```
public class SHA1 {
    private final int[] h = {
            0x67452301,
            0xEFCDAB89,
            0x98BADCFE,
            0x10325476,
            0xC3D2E1F0
        };
    private int[] int_digest = new int[5];
    private int[] tmp = new int[80];
    private byte[] ByteArrFormat(byte[] bytedata) {
        int zeros = 0;
        int size = 0;
        int n = bytedata.length;
        int m = n % 64;
        if (m < 56) {
            zeros = 55 - m;
            size = n - m + 64;
        } else if (m == 56) {
            zeros = 63;
            size = n + 8 + 64;
        } else {
            zeros = 63 - m + 56;
            size = (n + 64) - m + 64;
        }
        byte[] newbyte = new byte[size];
        for (int j = 0; j < n; j++){
            newbyte[j] = bytedata[j];
        }
        int l = n;
        newbyte[l++] = (byte) 0x80;
        for (int i = 0; i < zeros; i++) {
            newbyte[l++] = (byte) 0x00;
        }
        long N = (long) n * 8;
        byte h8 = (byte) (N & 0xFF);
        byte h7 = (byte) ((N >> 8) & 0xFF);
        byte h6 = (byte) ((N >> 16) & 0xFF);
        byte h5 = (byte) ((N >> 24) & 0xFF);
        byte h4 = (byte) ((N >> 32) & 0xFF);
        byte h3 = (byte) ((N >> 40) & 0xFF);
        byte h2 = (byte) ((N >> 48) & 0xFF);
        byte h1 = (byte) (N >> 56);
        newbyte[l++] = h1;
        newbyte[l++] = h2;
        newbyte[l++] = h3;
        newbyte[l++] = h4;
        newbyte[l++] = h5;
        newbyte[l++] = h6;
        newbyte[l++] = h7;
        newbyte[l++] = h8;
        return newbyte;
    }

    private int count1(int x, int y, int z) {
        return (x & y) | (~x & z);
    }

    private int count2(int x, int y, int z) {
        return x ^ y ^ z;
    }

    private int count3(int x, int y, int z) {
        return (x & y) | (x & z) | (y & z);
    }

    private int count3(int x, int y) {
        return (x << y) | x >>> (32 - y);
    }

    private int ByteArrToInt(byte[] bytedata, int i) {
        return    ((bytedata[i]   &   0xff)   <<   24)   |
((bytedata[i + 1] & 0xff) << 16) |
        ((bytedata[i + 2] & 0xff) << 8) | (bytedata[i +
3] & 0xff);
    }

    private  void  intToByteArray(int  intValue,  byte[]
byteData, int i) {
        byteData[i] = (byte) (intValue >>> 24);
        byteData[i + 1] = (byte) (intValue >>> 16);
        byteData[i + 2] = (byte) (intValue >>> 8);
        byteData[i + 3] = (byte) intValue;
    }

    public byte[] BytesToDigest(byte[] byteData) {
         for (int i = 0; i < int_digest.length; i++){
            int_digest[i] = h[i];
        }
        byte[] newbyte = ByteArrFormat(byteData);
        int MCount = newbyte.length / 64;
        for (int pos = 0; pos < MCount; pos++) {
            for (int j = 0; j < 16; j++) {
                tmp[j]   =   ByteArrToInt(newbyte,   (pos   *
64) + (j * 4));
            }
            for (int a = 16; a <= 79; a++) {
            tmp[a]  =  count3(tmp[a - 3]  ^  tmp[a - 8]  ^
tmp[a - 14] ^
                    tmp[a - 16], 1);
            }

            int[] tmpabcde = new int[5];

            for  (int  i1  =  0;  i1  <  tmpabcde.length;
i1++) {
                tmpabcde[i1] = int_digest[i1];
            }

            for (int j = 0; j <= 19; j++) {
                int tmp = count3(tmpabcde[0], 5) +
                    count1(tmpabcde[1],     tmpabcde[2],
tmpabcde[3]) + tmpabcde[4] +
                    this.tmp[j] + 0x5a827999;
                tmpabcde[4] = tmpabcde[3];
                tmpabcde[3] = tmpabcde[2];
                tmpabcde[2] = count3(tmpabcde[1], 30);
                tmpabcde[1] = tmpabcde[0];
                tmpabcde[0] = tmp;
```

```
            }

            for (int k = 20; k <= 39; k++) {
                int tmp = count3(tmpabcde[0], 5) +
                    count2(tmpabcde[1],     tmpabcde[2],
tmpabcde[3]) + tmpabcde[4] +
                    this.tmp[k] + 0x6ed9eba1;
                tmpabcde[4] = tmpabcde[3];
                tmpabcde[3] = tmpabcde[2];
                tmpabcde[2] = count3(tmpabcde[1], 30);
                tmpabcde[1] = tmpabcde[0];
                tmpabcde[0] = tmp;
            }

            for (int l = 40; l <= 59; l++) {
                int tmp = count3(tmpabcde[0], 5) +
                    count3(tmpabcde[1],     tmpabcde[2],
tmpabcde[3]) + tmpabcde[4] +
                    this.tmp[l] + 0x8f1bbcdc;
                tmpabcde[4] = tmpabcde[3];
                tmpabcde[3] = tmpabcde[2];
                tmpabcde[2] = count3(tmpabcde[1], 30);
                tmpabcde[1] = tmpabcde[0];
                tmpabcde[0] = tmp;
            }

            for (int m = 60; m <= 79; m++) {
                int tmp = count3(tmpabcde[0], 5) +
                    count2(tmpabcde[1],     tmpabcde[2],
tmpabcde[3]) + tmpabcde[4] +
                    this.tmp[m] + 0xca62c1d6;
                tmpabcde[4] = tmpabcde[3];
                tmpabcde[3] = tmpabcde[2];
                tmpabcde[2] = count3(tmpabcde[1], 30);
                tmpabcde[1] = tmpabcde[0];
                tmpabcde[0] = tmp;
            }

            for  (int  i2  =  0;  i2  <  tmpabcde.length;
i2++) {
                int_digest[i2]    =    int_digest[i2]    +
tmpabcde[i2];
            }

            for (int n = 0; n < tmp.length; n++) {
                tmp[n] = 0;
            }
        }
        byte[] digest = new byte[20];

        for (int i = 0; i < int_digest.length; i++) {
            intToByteArray(int_digest[i],  digest,  i  *
4);
        }

        return digest;
    }

    public String StringToDigest(byte[] byteData) {
        String strDigest = "";
        for      (int      i      =      0;      i      <
BytesToDigest(byteData).length; i++) {
            char[] Digit = {
                '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'a', 'b', 'c',
                'd', 'e', 'f'
                };
            char[] ob = new char[2];
            ob[0]   =   Digit[(BytesToDigest(byteData)[i]
>>> 4) & 0X0F];
            ob[1]  =  Digit[BytesToDigest(byteData)[i]  &
0X0F];

            String s = new String(ob);
            strDigest += s;

        }
        return strDigest;
    }
}
```

Untuk implementasi MD5, Mobile Hash menggunakan *library* MD5 yang terdapat di http://mobilepit.com/10/compact-md5-class-library-for-j2me-javame-app.html. Terdapat dua buah file kelas Java, yaitu MD5State. Java dan MD5.java yang saling berasosiasi.

## *B. Analisis Aplikasi Mobile Hash*

Telepon seluler yang mendukung aplikasi ini adalah yang berbasis MIDP 2.0. Tampilan dari hasil eksekusi aplikasi Mobile Hash adalah sebagai berikut.

Gambar di bawah ini adalah tampilan awal program. Untuk melanjutkan, pengguna harus memilih Proceed.

Phone number: 123456789

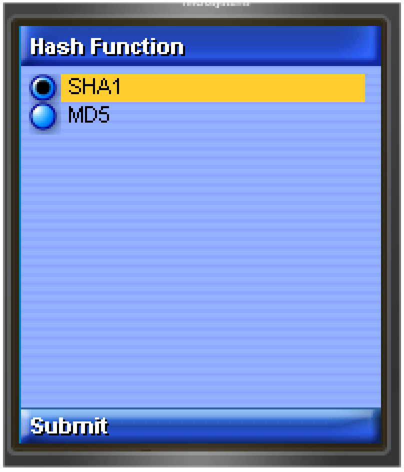
Digital Signature

WELCOME TO DIGITAL SIGNATURE

Vibration is off

**Gambar 5 Tampilan awal Mobile Hash**

Gambar di bawah ini adalah tampilan pemilihan fungsi *hash*. Pengguna dapat memilih SHA-1 atau MD5. Untuk kasus ini, SHA-1 dipilih.

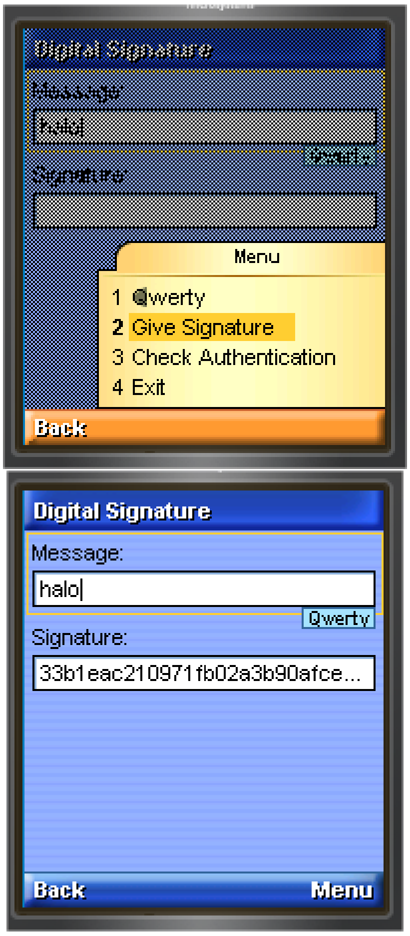


Hash Function

● SHA1
○ MD5

**Gambar 6 Tampilan pemilihan fungsi *hash***

Gambar di bawah ini adalah tampilan masukan teks dari pengguna. Setelah teks dimasukkan yaitu “halo”, pengguna memilih Give Signature. Lalu tanda tangan muncul pada *field* Signature.



Digital Signature

Message: halo

Signature:

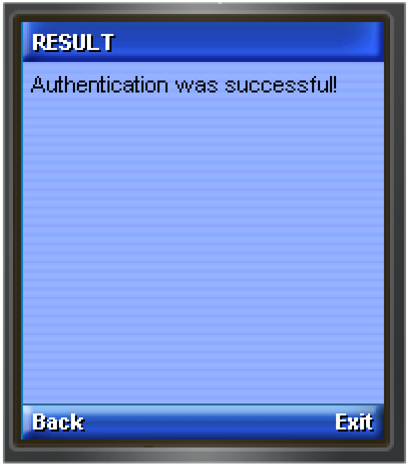
Digital Signature

Message: halo

Signature: 33b1eac210971fb02a3b90afce...

Gambar 7 Tampilan pemberian tanda tangan

Gambar di bawah ini adalah tampilan hasil setelah pengguna memilih Check Authentication pada menu.



RESULT

Authentication was successfull

Gambar 8 Tampilan hasil pengujian

Kemudian untuk menguji sensitivitas, teks pada *field* Message diubah, misalkan teks “halo” kita ubah menjadi “haloww”.

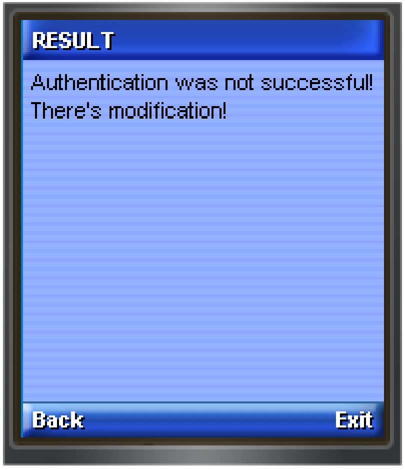
Digital Signature

Message: haloww

Signature: 33b1eac210971fb02a3b90afce...

Gambar 9 Tampilan pengubahan isi pesan

Setelah teks diubah, dilakukan pengujian otentikasi. Hasil yang didapat adalah sebagai berikut.

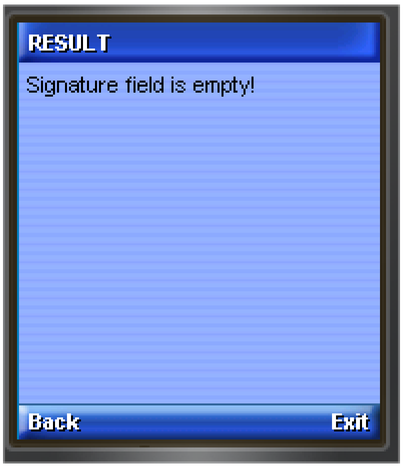


RESULT

Authentication was not successfull
There's modification!

Gambar 10 Tampilan hasil pengujian jika isi data dengan nilai *hash* tidak cocok

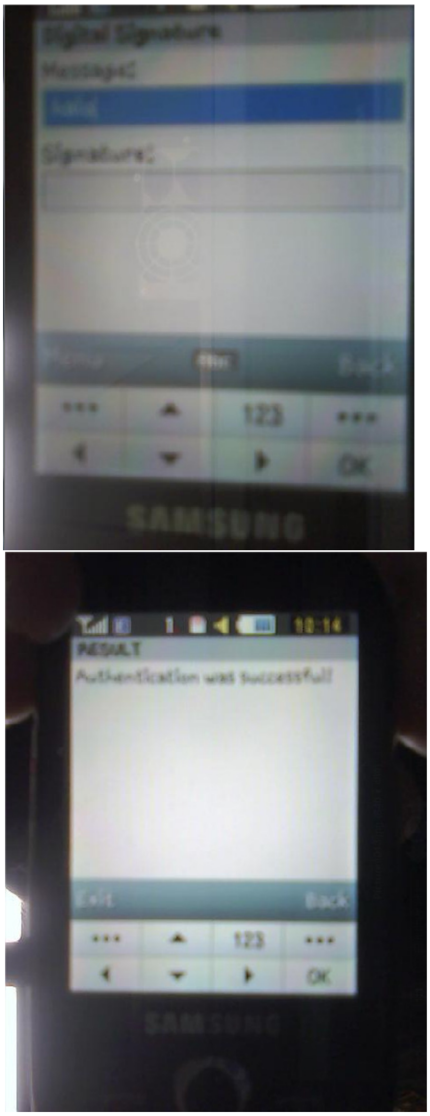
Selain itu, aplikasi ini juga menangani ketiadaan tanda tangan dalam dokumen jika ingin menguji otentikasi. Hal ini terjadi jika *field* Signature kosong.



Signature field is empty!

**Gambar 11 Tampilan hasil jika tidak ada tanda tangan untuk pengecekan**

Aplikasi Mobile Hash ini dicoba ke ponsel Samsung GT S3653. Hasilnya adalah sebagai berikut.



Authentication was successfull!

**Gambar 12 Tampilan Mobile Hash di posel Samsung GT S3653**

Tampilan aplikasi Mobile Hash ini berbeda-beda tergantung jenis ponsel. Untuk melakukan fungsinya, kedua pihak yang melakukan pertukaran data harus memiliki aplikasi ini.

Kekurangan aplikasi ini adalah tidak adanya fasilitas untuk melakukan pengambilan data dari memori telepon. Jadi data harus dimasukkan secara manual, baik teks maupun tanda tangannya. Namun hal ini bisa dipermudah jika ponsel menyediakan fitur salin teks (*copy-paste*), sehingga data yang dimasukkan tidak perlu diketik satu per satu.

## V. KESIMPULAN

Masalah keamanan pertukaran data pada telepon seluler menjadi hal yang penting di zaman sekarang. Salah satunya adalah masalah keaslian dan otentikasi data. Salah satu solusi untuk mengatasi masalah keamanan ini adalah menggunakan fungsi *hash* satu arah. Fungsi *hash* dapat digunakan untuk mengamankan data pada ponsel, salah satunya dengan membuat aplikasi yang menggunakan fungsi *hash* pada ponsel dengan menggunakan bahasa Java.

## REFERENSI

A.Menezes, P.van Oorschot,dan S.Vanstone. “Applied Cryptography”. CRCPress,1996, bab 1 & 9.

Munir,Rinaldi.”Diktat Kuliah Kriptografi”. Bandung: Depatemen Teknik Informatika ITB,2005, halaman 152-172.

http://home.comcast.net/~bretm/hash/

http://kiospaktani.com/mod.php?mod=publisher&op=viewarticle&artid=150

http://mobilepit.com/10/compact-md5-class-library-for-j2me-javame-app.html

http://tools.ietf.org/html/rfc1321

http://w2.eff.org/Privacy/Digital_signature/?f=fips_sha_shs.standard.txt

http://www.faqs.org/rfcs/rfc3174.html

http://www.java-tips.org/java-me-tips/midp/sending-receiving-sms-on-j2me-device-3.html

http://www.puputs.com/2008/06/tutorial-membuat-program-java-j2medi.html

http://www.roseindia.net/j2me/

http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

http://www.schneier.com/blog/archives/2005/11/nist_hash_works_4.html

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 16 Mei 2010

Juliana Amytianty Kombaitan
13507068